Christu Jyothi Institute of Technology & Science

JANGAON

COMPUTER SCIECNE ENGINEERING DEPARTMENT

B.TECH – III YEAR II SEMESTER

(Regulation: R22)

(2025-2026)

FULL STACK DEVELOPMENT LAB MANUAL

(CS611PE)

Prepared By: Lagishetti Suresh *M.Tech(CSE) – CSE, CJITS*

List of Experiments

1. Create an application to setup node JS environment and display “Hello World”.

2. Create a Node JS application for user login system.

3. Write a Node JS program to perform read, write and other operations on a file.

4. Write a Node JS program to read form data from query string and generate response using NodeJS

5. Create a food delivery website where users can order food from a particular restaurant listed in the website for handling http requests and responses using NodeJS.

6. Implement a program with basic commands on databases and collections using MongoDB.

7. Implement CRUD operations on the given dataset using MongoDB.

8. Perform Count, Limit, Sort, and Skip operations on the given collections using MongoDB.

9. Develop an angular JS form to apply CSS and Events.

10. Develop a Job Registration form and validate it using angular JS.

11. Write an angular JS application to access JSON file data of an employee from a server using $http service.

12. Develop a web application to manage student information using Express and Angular JS.

13. Write a program to create a simple calculator Application using React JS.

14. Write a program to create a voting application using React JS

15. Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days using react application.

16. Build a music store application using react components and provide routing among the pages.

17. Create a react application for an online store which consist of registration, login, product information pages and implement routing to navigate through these pages.

EXPERIMENT – 1

1. Create an application to setup node JS environment and display "Hello World".

AIM: To set up the Node.js development environment and create a simple application that displays **"Hello World"** on the console and/or browser.

OBJECTIVE:

- To install and configure Node.js.
- To initialize a Node.js project.
- To write a basic JavaScript program using Node.js.
- To execute a Node program in the terminal.
- To create a simple HTTP server (optional).

THEORY:

Node.js is an open-source, cross-platform runtime environment built on Chrome's V8 JavaScript engine. It allows developers to execute JavaScript code outside the browser. Node.js is widely used for backend development, API creation, and real-time applications.

Key points:

- Uses asynchronous & event-driven architecture
- High performance due to V8 engine
- Supports large ecosystem (NPM modules)

PROCEDURE

Step 1: Install Node.js (first you need to install Python in the system).

Type the command at command prompt: **C:\> pip install nodejs-bin (After Install check below commands)**

C:\> node –v

C:> npm -v

Step 2: Create Project Folder

C:\> md exp1

C:\> cd exp1

Step 3: Initialize Node Project

npm init -y

Step 4: Create `app.js` File

Write the following code: (Notepad or visual Studio IDE)

```
const http = require("http");

const server = http.createServer((req, res) => {
   res.writeHead(200, { "Content-Type": "text/plain" });
   res.end("Hello World");
});

server.listen(3000, () => {
   console.log("Server running at http://localhost:3000");
});
```

Step 5: Run the Application (Goto the command prompt at same directory)

Type following command:

C:\exp1> node app.js

Server running at http://localhost:3000

(Copy given URL at Browser Address Bar for Output).

EXPERIMENT – 2

2. Create a Node JS application for user login system.

AIM:
To create a simple Node.js application that connects to MySQL to register a user (with hashed password) and allow login verification.

OBJECTIVE

- Connect Node.js app with MySQL database.
- Create `users` table.

THEORY (brief)

Node.js allows creating servers using the built-in **`http`** module. The **MySQL2** module enables Node.js to connect to a MySQL database.

PROCEDURE:

1. Create project folder and initialize:

```
C:\> md exp2
C:\>cd exp2
C:\exp2>npm init -y
```

Process:

1. Browser sends login form data (POST request).
2. Node.js server reads the request body manually using events (`data` and `end`).
3. The `querystring` module converts the POST data into key-value format.
4. A SQL query checks whether username & password exist in the MySQL `users` table.
5. The server responds with either *Login Successful* or *Invalid Credentials*.

This experiment shows how backend validation works without frameworks.

PREPARATION / SQL (run in MySQL):

```
CREATE DATABASE IF NOT EXISTS lab_auth;
USE lab_auth;

CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

PROCEDURE

1. Install Node.js and MySQL on the system.
2. Create project folder and files:
   - `server.js`
   - `login.html`
   - `styles.css`
3. Install MySQL module:

   C:\> npm install mysql2

4. Start MySQL and create table using SQL above.
5. Write code in `server.js` to:

   - Connect to database
   - Serve HTML and CSS files
   - Receive POST request
   - Validate credentials

6. Run server:

   C:\exp2>node server.js

7. Open browser:

   http://localhost:3000

PROGRAM(CODE):

Server.js:

```
const http = require("http");
const fs = require("fs");
const querystring = require("querystring");
const mysql = require("mysql2");

// MySQL connection
const db = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "",
   database: "exp2"
});

db.connect((err) => {
   if (err) throw err;
   console.log("Connected to MySQL");
});

const server = http.createServer((req, res) => {

   if (req.method === "GET" && req.url === "/") {
      fs.readFile("login.html", (err, data) => {
         res.writeHead(200, { "Content-Type": "text/html" });
         res.end(data);
      });
```

```
    }

    else if (req.method === "GET" && req.url === "/styles.css") {
        fs.readFile("styles.css", (err, data) => {
            res.writeHead(200, { "Content-Type": "text/css" });
            res.end(data);
        });
    }

    else if (req.method === "POST" && req.url === "/login") {
        let body = "";

        req.on("data", chunk => {
            body += chunk.toString();
        });

        req.on("end", () => {
            const { username, password } = querystring.parse(body);

            const sql = "SELECT * FROM users WHERE username = ? AND password = ? LIMIT 1";

            db.query(sql, [username, password], (err, results) => {
                if (err) throw err;

                res.writeHead(200, { "Content-Type": "text/html" });

                if (results.length > 0) {
                    res.end(`<h2 style="color:green;text-align:center;">Login Successful! Welcome
${username}</h2>`);
                } else {
                    res.end(`<h2 style="color:red;text-align:center;">Invalid Username or Password</h2>`);
                }
            });
        });
    }

    else {
        res.writeHead(404, { "Content-Type": "text/plain" });
        res.end("404 Not Found");
    }
});

server.listen(3000, () => {
    console.log("Server running at http://localhost:3000");
});
```

index.html :

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Simple Login</title>
    <link rel="stylesheet" href="styles.css">
```

```
</head>
<body>
<div class="container">
    <h2>Login</h2>
    <form method="POST" action="/login">
        <input type="text" name="username" placeholder="Enter Username" required>
        <input type="password" name="password" placeholder="Enter Password" required>
        <button type="submit">Login</button>
    </form>

    <p id="message"></p>
</div>
</body>
</html>
```

styles.css :

```
body {
    font-family: Arial, sans-serif;
    background: #f4f4f4;
    display: flex;
    height: 100vh;
    align-items: center;
    justify-content: center;
}
.container {
    background: white;
    padding: 25px;
    border-radius: 10px;
    width: 300px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}
h2 {
    text-align: center;
}
input {
    width: 100%;
    padding: 10px;
    margin: 8px 0;
    border-radius: 5px;
    border: 1px solid #aaa;
}
button {
    width: 100%;
    padding: 10px;
    background: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
button:hover {
    background: #0056b3;
}
```

EXPERIMENT – 3

Write a Node JS program to perform read, write and other operations on a file.

AIM: To write a Node.js program that performs **write**, **read**, **append**, **rename**, and **delete** operations on a file using the built-in `fs` module.

OBJECTIVE:

- To understand the use of Node.js **File System (fs)** module.
- To create, read, update, rename, and delete files programmatically.
- To learn asynchronous callback-based file handling.

THEORY:

Node.js provides the **fs (File System)** module to work with files.
It allows both **synchronous** and **asynchronous** operations such as:

- `writeFile()` → Create/write file
- `readFile()` → Read file
- `appendFile()` → Add data to existing file
- `rename()` → Change file name
- `unlink()` → Delete a file

In this experiment, we use **asynchronous** functions because they do not block the main thread.

PROGRAM:
file_operations.js

```
const fs = require("fs");

// 1. Write data to a file
fs.writeFile("sample.txt", "This is the initial content.\n", (err) => {
   if (err) throw err;
   console.log("File created and data written!");

   // 2. Read the file
   fs.readFile("sample.txt", "utf8", (err, data) => {
      if (err) throw err;
      console.log("File Content:");
      console.log(data);

      // 3. Append data to the file
      fs.appendFile("sample.txt", "This line is appended.\n", (err) => {
         if (err) throw err;
         console.log("Data appended to file!");

         // 4. Rename the file
         fs.rename("sample.txt", "new_sample.txt", (err) => {
            if (err) throw err;
            console.log("File renamed to new_sample.txt");
```

```
            // 5. Delete the file
            fs.unlink("new_sample.txt", (err) => {
                if (err) throw err;
                console.log("File deleted successfully!");
            });
        });
    });
});
```

Run Procedure (Output):

```
C:\> md exp3
C:\exp3>node file_operations.js
File created and data written!
File Content:
This is the initial content.

Data appended to file!
File renamed to new_sample.txt
File deleted successfully!
```

EXPERIMENT – 4

Write a Node JS program to read form data from query string and generate response using NodeJS

AIM

To write a Node.js program that reads form data sent through a **query string** and generates an appropriate response.

OBJECTIVE

- To understand how Node.js handles HTTP GET requests.
- To extract parameters from a URL query string.
- To generate dynamic output based on user input.

THEORY

In HTTP GET requests, form data is sent in the URL after a `?` symbol in key–value pairs.
Example:

http://localhost:3000/?name=John&age=20

Node.js provides the built-in **url** module to parse query strings:

- `url.parse(req.url, true).query` gives an object with form data.

PROGRAM:

server.js

```
const http = require("http");
const url = require("url");

const server = http.createServer((req, res) => {

  // Parse URL and get query string data
  const q = url.parse(req.url, true).query;

  // If no data, show form
  if (!q.name && !q.age) {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.end(`
            <center><div>
      <h2>Enter Your Details</h2>
      <form method="GET" action="/">
        <input type="text" name="name" placeholder="Enter Name" required><br><br>
        <input type="number" name="age" placeholder="Enter Age" required><br><br>
        <button type="submit">Submit</button>
      </form></div></center>
    `);
  }
```

```
    else {
        // Data available → generate response
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end(`
            <h2>Form Submitted Successfully!</h2>
            <p><b>Name:</b> ${q.name}</p>
            <p><b>Age:</b> ${q.age}</p>
        `);
    }
});

// Start server
server.listen(3000, () => {
    console.log("Server running at http://localhost:3000");
});
```

OUTPUT

Case 1: Initial Page

Displays form:

After submitting

Example:
URL → `http://localhost:3000/?name=mahesh&age=25`

EXPERIMENT – 5

Create a food delivery website where users can order food from a particular restaurant listed in the website for handling http requests and responses using NodeJS.

AIM:

To create a simple food delivery website using Node.js where restaurant names, menu items, and prices are stored in a JSON file**, and to handle HTTP requests and responses.**

OBJECTIVE

- To understand HTTP request and response handling in Node.js
- To create a basic web server using Node.js
- To display restaurant and food items
- To process user order input

THEORY

Node.js can read external files using the **fs module**: In this experiment:

- Restaurant and menu details are stored in a **JSON file**
- Node.js reads the JSON file and displays restaurants and menus
- User selects food items
- Server calculates the total bill and sends response
- `styles.css` is used for styling the pages

PROCEDURE:

- Create files: `server.js`, `data.json`, `styles.css`
- Run:

C:\exp5> node server.js

PROGRAM:

data.json:

```
{
 "restaurants": [
   {
    "id": "spicyhub",
    "name": "Spicy Hub",
    "menu": {
      "Biryani": 200,
      "Paneer": 150,
      "Roti": 20
    }
```

```
    },
    {
     "id": "italiandelight",
     "name": "Italian Delight",
     "menu": {
      "Pizza": 250,
      "Pasta": 180,
      "Garlic Bread": 120
     }
    }
  ]
}
```

server.js

```
const http = require("http");
const fs = require("fs");
const querystring = require("querystring");

// Read JSON data
const data = JSON.parse(fs.readFileSync("data.json", "utf8"));

const server = http.createServer((req, res) => {

   // Serve CSS
   if (req.url === "/styles.css") {
      fs.readFile("styles.css", (err, css) => {
         res.writeHead(200, { "Content-Type": "text/css" });
         res.end(css);
      });
   }

   // Home Page - Restaurant List
   else if (req.url === "/" && req.method === "GET") {
      let restHtml = "";
      data.restaurants.forEach(r => {
         restHtml += `<a href="/menu?rest=${r.id}" class="btn">${r.name}</a><br><br>`;
      });

      res.writeHead(200, { "Content-Type": "text/html" });
      res.end(`
         <html>
         <head>
            <title>Food Delivery</title>
            <link rel="stylesheet" href="styles.css">
         </head>
         <body>
            <div class="container">
               <h2>Food Delivery Website</h2>
               <h3>Select Restaurant</h3>
               ${restHtml}
            </div>
         </body>
         </html>
      `);
```

```
}

// Menu Page
else if (req.url.startsWith("/menu") && req.method === "GET") {
   const query = new URL(req.url, "http://localhost").searchParams;
   const restId = query.get("rest");
   const restaurant = data.restaurants.find(r => r.id === restId);

   if (!restaurant) {
      res.writeHead(404, { "Content-Type": "text/html" });
      res.end("<h3>Restaurant not found</h3>");
      return;
   }

   let menuHtml = "";
   for (let item in restaurant.menu) {
      menuHtml += `
         <input type="checkbox" name="food" value="${item}">
         ${item} (Rs. ${restaurant.menu[item]})<br>
      `;
   }

   res.writeHead(200, { "Content-Type": "text/html" });
   res.end(`
      <html>
      <head>
         <title>${restaurant.name}</title>
         <link rel="stylesheet" href="styles.css">
      </head>
      <body>
         <div class="container">
            <h2>${restaurant.name} - Menu</h2>
            <form method="POST" action="/order?rest=${restaurant.id}">
               ${menuHtml}<br>
               <button type="submit" class="btn">Place Order</button>
            </form>
            <br>
            <a href="/" class="link">Back</a>
         </div>
      </body>
      </html>
   `);
}

// Order Confirmation Page
else if (req.url.startsWith("/order") && req.method === "POST") {
   const query = new URL(req.url, "http://localhost").searchParams;
   const restId = query.get("rest");
   const restaurant = data.restaurants.find(r => r.id === restId);

   let body = "";
   req.on("data", chunk => body += chunk.toString());

   req.on("end", () => {
```

```
        let formData = querystring.parse(body);
        let items = formData.food || [];

        if (!Array.isArray(items)) items = [items];

        let total = 0;
        items.forEach(item => {
            total += restaurant.menu[item];
        });

        res.writeHead(200, { "Content-Type": "text/html" });
        res.end(`
            <html>
            <head>
                <title>Order Confirmation</title>
                <link rel="stylesheet" href="styles.css">
            </head>
            <body>
                <div class="container">
                    <h2>Order Confirmed</h2>
                    <p><b>Restaurant:</b> ${restaurant.name}</p>
                    <p><b>Items Ordered:</b> ${items.join(", ")}</p>
                    <p><b>Total Amount to Pay:</b> ${total}</p>
                    <p class="success">Thank you for your order!</p>
                    <a href="/" class="btn">Home</a>
                </div>
            </body>
            </html>
        `);
    });
  }

  // Invalid URL
  else {
    res.writeHead(404, { "Content-Type": "text/html" });
    res.end("<h3>404 - Page Not Found</h3>");
  }
});

// Start Server
server.listen(3000, () => {
  console.log("Server running at http://localhost:3000");
});
```

EXPERIMENT – 6

Implement a program with basic commands on databases and collections using MongoDB.

AIM:

To implement a Node.js program that performs **basic MongoDB commands** such as creating a database, creating a collection, inserting documents, reading documents, updating documents, and deleting documents.

OBJECTIVE

- To understand MongoDB database and collections
- To perform CRUD operations using MongoDB
- To connect Node.js with MongoDB
- To execute basic MongoDB commands programmatically

THEORY

MongoDB is a **NoSQL document-oriented database** that stores data in JSON-like documents.
In MongoDB:

- A **database** contains **collections**
- A **collection** contains **documents**
- CRUD operations include:
  - **Create** → `insertOne()`
  - **Read** → `find()`
  - **Update** → `updateOne()`
  - **Delete** → `deleteOne()`

Node.js connects to MongoDB using the official **mongodb** driver.

PROCEDURE

1. Start MongoDB server
2. Create a Node.js project folder
3. Install MongoDB driver
4. Write Node.js program to perform CRUD operations
5. Run the program and observe output

INSTALLATION

1. Create a exp6 folder.
2. Change to exp6 folder
3. Type the below command to Install MongoDB driver using npm in current Directory

   C:\exp6> npm install mongodb

PROGRAM:

Create a code file: mongo_basic.js

Program Code:

```
const { MongoClient } = require("mongodb");
// MongoDB URL
const url = "mongodb://localhost:27017";
const client = new MongoClient(url);
// Database and Collection names
const dbName = "collegeDB";
const collectionName = "students";
async function run() {
  try {
    // Connect to MongoDB
    await client.connect();
    console.log("Connected to MongoDB");
    const db = client.db(dbName);
    const collection = db.collection(collectionName);
    // 1. Insert Document
    const insertResult = await collection.insertOne({
      rollno: 101,
      name: "Ravi",
      branch: "CSE",
      marks: 85
    });
    console.log("Document Inserted");
    // 2. Read Documents
    const students = await collection.find({}).toArray();
    console.log("Documents in collection:");
    console.log(students);
    // 3. Update Document
    const updateResult = await collection.updateOne(
```

```
        { rollno: 101 },
        { $set: { marks: 90 } }
      );
      console.log("Document Updated");
      // 4. Delete Document
      const deleteResult = await collection.deleteOne({ rollno: 101 });
      console.log("Document Deleted");
    } catch (err) {
      console.error(err);
    } finally {
      // Close connection
      await client.close();
      console.log("Connection Closed");
    }
}
run();
```

RUN:

1. Start MongoDB server: